Babbage

Version 0.92

USER'S MANUAL

Contents

Preface

1. What is Babbage
  a) The control unit
  b) The arithmetic unit
  c) The memory unit
  d) The Input/Output unit
  e) The Bus
  f) Remote control
  g) Toolbar

2. How a computer works
  a) Digital and electronics
  b) The fetch execute cycle

3. What instructions can Babbage carry out?
4. How to enter programs into babbage
5. How to run programs
6. Saving and Loading programs

7. Some example programs
  allcodes.bab
  lights on.bab
  youdecide.bab
  cycling.bab
  square a number.bab

8. Appendices
  Babbage instruction set
  Limitations in Babbage
  Some online resources about Computing
  Blank Babbage programming sheet

Preface

This guide has been written to help you use Babbage, it isn't a textbook or course book for computing or computer science.

Babbage has been written to help students (of all ages) to understand the beauty and elegance of the inside of real computers. Almost everything in our lives now contains either a full computer or at least a microprocessor that helps that thing have more intelligence. Anyone under 30 years of age probably cannot remember a time when computers weren't everywhere, but that makes it more important that we all understand the basics of a computer and don't just treat them as magic boxes.

We hope that by the end of your use of Babbage you understand why a good definition of a computer might be a fast idiot.

Babbage can be used by anyone. It has uses in primary (KS2) and secondary schools (KS3 and KS4). The full use of the instruction set and full understanding of all of the operations is best suited to KS4 and GCSE courses, but it is still useful for lower down the years. Similarly, to help with Advanced level studies you really need to think about the "binary" inside a real computer, but Babbage still helps understand the principles. (note that an A level version is also being prepared).

As far as we know nothing in the guide tells you incorrect ideas about computers, but we haven't got the space, nor is it our job, to try to copy all of the good stuff in textbooks, online sites etc. Appendix 3 in the guide does point you towards some of the better online sources.

We hope you enjoy using Babbage and find time to visit the **kilkiesoft** website (www.kilkiesoft.com) and leave comments and suggestions for improvements. We would also recommend that teachers join Computers at School group and join in the task of creating teaching and learning resources to share.

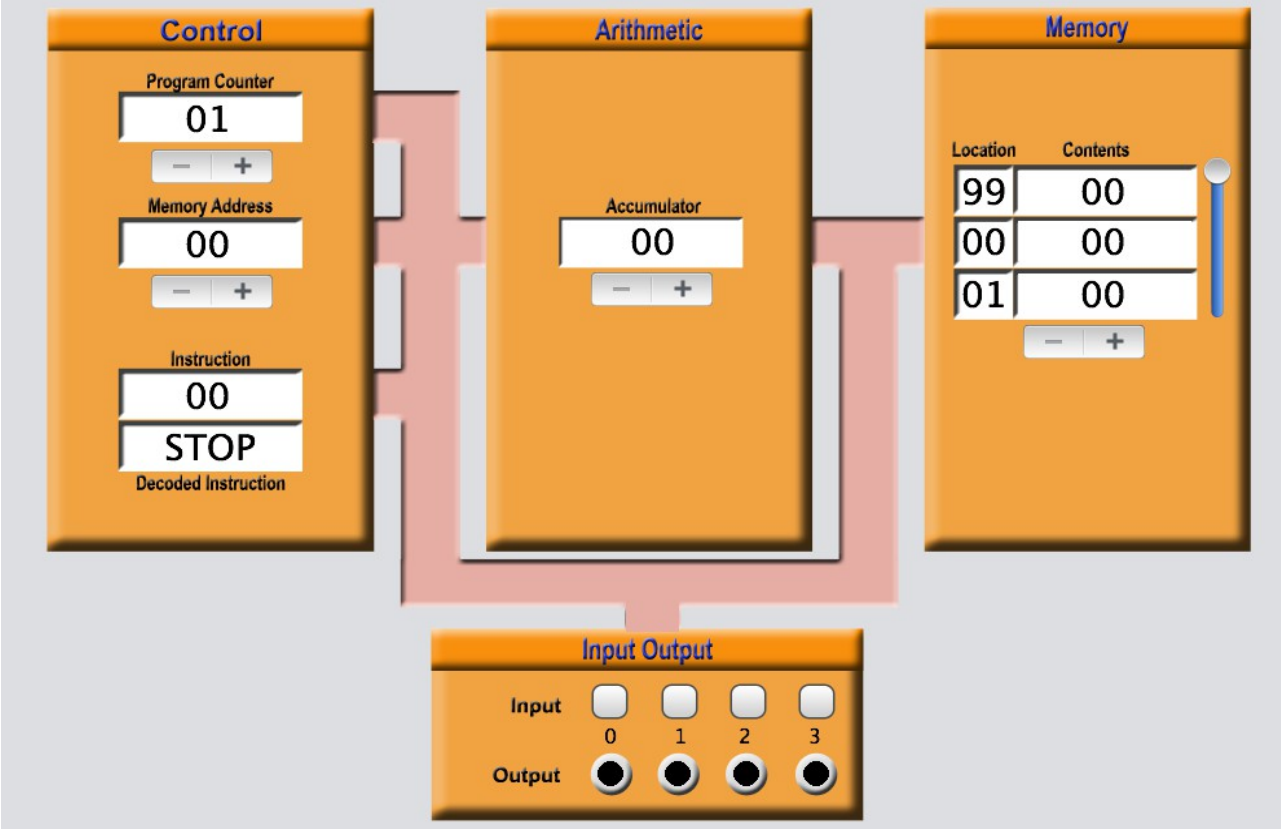
Martin Kilkie, Kilkiesoft
December 2013

What is Babbage?

Babbage is a model computer, that is it contains all the main parts that a real computer contains but is simplified so that you can see the parts and what they do when it is working.

Some definitions of a computer would call it a CPU (central processing unit) and other people would say that it is a whole computer as it contains a CPU, memory and input and output devices. Modern computers, tablets, phones etc. use components that make the difference between a CPU (like an ARM chip or an Intel i4 etc.) and a whole computer difficult to distinguish. For example in the Raspberry Pi the whole computer fits is built around one chip.

Babbage is shown below:



This is MUCH simpler than even the simplest real CPU/computer but it contains all the parts that are needed. It also has some parts that are not in a real computer and we will explain these later in this section.

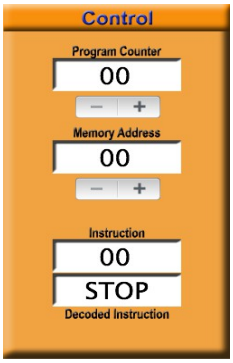
As we go through the parts we will explain how Babbage differs from real computers.

You can see from the diagram that there are 4 main parts (called units) and a route that connects the units together (this is called a bus).

The 4 units are: Control Unit, Arithmetic Unit, Memory Unit and the Input and Output Unit (oh and the bus of course!)

The Control Unit

This unit of Babbage contains all of the “intelligence” that instructs the other units to carry out tasks. It contains 3 registers (boxes that can hold numbers) and one extra box that isn’t present in a real computer.



Box	Purpose
Program Counter	This register is used to keep track of which instruction in memory is next to be fetched. So, if it contains 05 it means that the instruction in memory location 05 contains the next instruction.
Memory Address	This register controls which location in memory is to be read or written to. So, if it contains 04 then memory location 04 can be read (copied to somewhere else) or written (replaced with a number from somewhere else).
Instruction	This register receives a number from the memory
Decoded Instruction	This additional register in Babbage doesn’t exist in a real computer and is there so that we (humans) can see what the instruction in the instruction register is, without looking it up in the manual.

You can change only two of the registers in the Control unit, the **Program Counter** and the **Memory Address**. The **Instruction Register** only changes when it receives a new instruction from memory, you cannot change it or the decode instruction register manually.

Both the **Program Counter** and **Memory Address** registers have a “**stepper**” (the plus/minus button below them.) This stepper changes the value of the register either up or down.

You will notice that all the controls in Babbage, when you get to 99 and press **+** it goes back to zero and when you get to 00 and press **–** it goes back to 99. Think of it as a revolving wheel, there is nothing above 99 and nothing below 00, so we choose to just connect the ends to make control easier.

Notice that when you change the Memory Address register this alters the window in the memory that you are looking at. This isn’t quite how it should work, but it makes it easier to control Babbage. All of the Babbage operations are unaffected by this deliberate decision to work a bit differently.

The Arithmetic Unit

This unit carries out all of the “arithmetic” in Babbage. It can also carry out “logic” instructions (true and false) and in real computers the unit is called the Arithmetic and Logic Unit or ALU. Babbage only contains one box (register) in its arithmetic unit, while real computers have other boxes called general registers that are used to store numbers and move them around. This is because moving them around the arithmetic unit is much faster than to and from the memory (which you will remember is usually outside the main CPU chip). In many if not most modern computers there are several accumulators and in graphics units there can be hundreds of “parallel” accumulators.

Babbage's accumulator can only carry out addition and subtraction instructions, whereas most modern computers can do more complex multiplication and division. You might remember from your primary school that multiplication and division are really lots of additions or subtractions anyway.

	Box	Purpose
	Accumulator	The accumulator is it the calculator in a computer. Only numbers in this box can be added to or subtracted. Other numbers need to be copied to the accumulator by using other instructions and then moved back after the arithmetic

There is only one register in the arithmetic unit and this is changed using the stepper (the plus/minus button below it).

As with the other registers in Babbage you can only alter the Accumulator between 00 and 99.

Note that arithmetic in Babbage will be affected by this and you will need to remember this in any programs you write. Real computers can handle much (MUCH) bigger numbers nowadays but the problem of getting an answer bigger than it can cope with still exists. This version of Babbage doesn’t tell you when it goes above 99 or below 00 but real ones do with Flags.

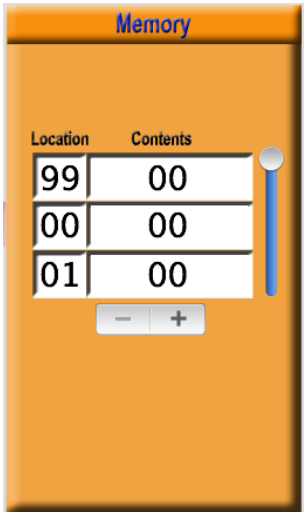
The Memory Unit

Memory is the name given to the place where information is stored. One of the standard things about all modern computers is that the memory is used for two different purposes:

1. The numbers that make up the instructions for the computer are stored in memory.
2. Any data used by the computer is also stored in the memory.

Real computers contain Gigabytes (109 bytes, but usually 1000 Mbytes) but poor old Babbage for simplicity only contains 100 (0 to 99) locations (places to put numbers). This 100 location memory is not very big but you can write quite complex programs in 100 locations. Remember that you also have to put any numbers you want to work on (data) into the memory too.

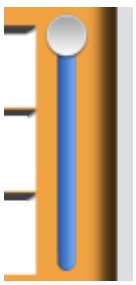
The reason for the 0-99 limit is quite simple, we wanted to keep the number of digits to 2 digits in all the registers. No real reason except to keep things simple. Note that this will have an impact that you will have to remember if you do arithmetic that adds up to more than 99!!



Box	Purpose
There are three memory locations shown in the memory unit.	This is a 3 location “window” into the 100 locations you can use. The location the computer wants to read or write is always the middle one and this is the one pointed to by the Memory Address register. The memory is used to store both program instructions and data for the programs. It is up to you to remember which is which and where you keep them. For any A level students that use the stack you will need to keep 80-99 separate too for the stack.

The memory unit has two different controls that work with it. The only location that you can alter the contents of is the middle one and this is altered using the stepper, again only between 00 and 99.

To alter which memory location is in the middle you can use either the stepper on the **Memory Address** register (in the Control unit) or to change it faster you can use the slider on the Memory Unit itself. Note that when you change the slider you change the **Memory Address** register at the same time.

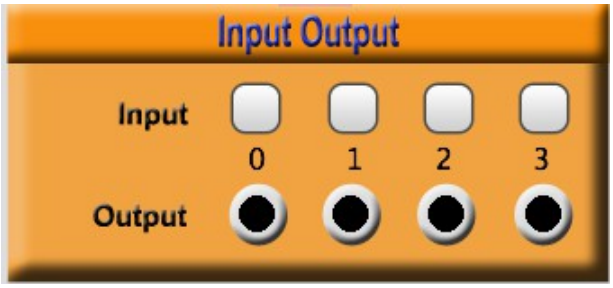


The Input Output Unit

As you probably know inputs are things that come in from outside the computer and outputs are signals that the computer sends out of the computer.

In the real world these can be as obvious as: Input (keyboard, mouse, finger, scanner), Output (printer, speakers for music, etc.) for general computers and tablets, industrial uses of computers make this much wider for example in a washing machine the computer inputs might be water temperature, weight of the load etc. and outputs might be opening and closing hot and cold taps, spinning the drum etc.

Babbage's input and output unit is probably the simplest part of Babbage. Although you might at first just turn on lights that have had their switches turned on, this is the same principle as using a computer to control an aircraft or make a coffee percolator work.



Box	Purpose
Input 0,1,2,3	These inputs can be switched (turned on) by touching. Babbage can then “read” whether they are off or on using the correct instructions and can then do something as a result. Note that program instructions cannot change the switches only read them (as with real Inputs).
Output 0,1,2,3	These can only be turned on by Babbage instructions not manually. Also you cannot tell from an instruction whether the output is on or off, you have to do that with your program!

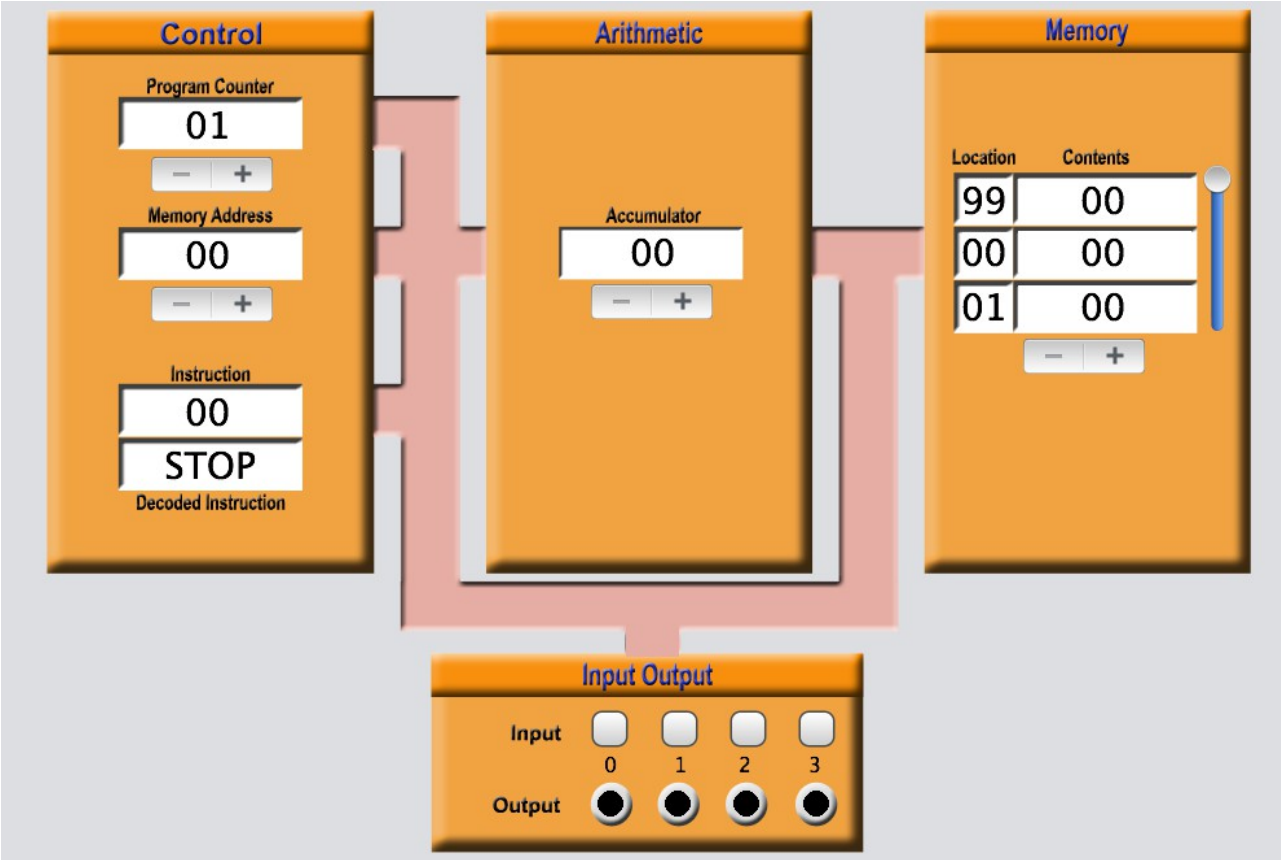
The Babbage Bus

The bus is the standard pathway that connects the different parts of the computer. In real computers the idea of serial and parallel is important. Also in real computers there is often one bus for changing the memory location and another bus takes numbers to and from the memory and other unit's registers.

The reason they use two buses is usually that there is a lot of memory (remember Gigabytes) and it needs to be able to label these big numbers, whereas the numbers in the registers is “usually” smaller. All of this is related to 32 bit and 64 bit computers that you might see advertised.

Babbage only has one combined address bus and data bus. This is to make the diagram simpler to follow and to let us keep all the numbers involved in the computer two digits.

The “pink” roadway on Babbage is the address/data bus.



The Babbage Remote

Real computers are quite complex and except for devices such as the Raspberry Pi you rarely see the insides and certainly cannot alter them directly. Earlier computers had what was called a “Front Panel” which let the user of the computer alter the memory and register but this isn’t usually possible today.

Babbage needs some controls and we invented the Remote control to do this. The buttons on the remote are quite simple, but we give details anyway (it fills up a bit of space!)

You will notice that buttons on the Babbage remote appear and disappear from time to time. This is deliberate and the appearances and disappearances makes sense when you are using Babbage. For example you can only “**stop**” a program when it is running and you can only **execute** an instruction once you have fetched it and you must **execute** the last instruction before you can **fetch** the next one.

Button	**Purpose**
Fetch	Tells Babbage to fetch the next instruction from memory. The next instruction is the one in the location stored in the Program Counter.
Execute	This button tells Babbage to carry out the instruction that has been fetched from memory and is in the Instruction register.
Run	This tells Babbage to run a whole program and only stop when it reaches an instruction to stop.
Stop	This stops a Babbage program which is being carried out, either because you get bored or your program is stuck in a loop.
Rewind	This rewinds the Program Counter to 00 and lets you restart your program.
Run speed	This slider lets you alter the speed that Run carried out instructions at. Note that it does not alter the speed of fetch and execute.

Notice that although rewind goes back to 00 you can run programs from any location or fetch and execute any instruction.

The Babbage Toolbar

In addition to telling Babbage what to do with the Remote control there are several other obvious things you will want to do with Babbage and these are all shown in the Toolbar.

When you load or save a program the name of the program will also show on the toolbar, but usually it looks as shown above.

Button	Purpose
Sound	On or Off toggle that either plays beeps when fetching and executing instructions or turns them off when you get bored by the noises.
Wastebin	This clears the current “program”, i.e. the contents of Babbage memory (all of it) and resets things back to the start. Use rewind if you just want to go back to the start of the program (you have been warned).
Load	This loads a program saved earlier, or one of the example programs. You can also delete program here (not the example ones).
Save	This lets you give a name and write a description of the programs you write then save them in your ipad for later.
Information	This button gives some help with Babbage and also tells you a bit about the program version and us.

In the release version of Babbage we will allow connections to your Dropbox account (and perhaps other cloud thingies).

2. How a computer works

Babbage is a tool to help you understand real computers, but it is not real itself. Although we said this was not a textbook we feel that a little bit of the theory behind real computers is needed to get you started and also to explain where and why some things in Babbage are a bit different.

Digital

All (well most) modern computers are digital. When we say most this is because there is some exciting work with “analogue” computers and also some “quantum computers” that work differently than most.

Digital just means numbers and this is one of the key things about computers. They only understand numbers, not words, not pictures, not videos and not sound. All they can do is move numbers around and do some quite basic arithmetic. To get from this simple numbers to the pictures, video and sound on your iPads, Macs , PCs, phones and tablets (IOS and Android) you need clever humans to write very long complex programs that turn the simple numbers and arithmetic into the complex “software” “apps” we expect.

Electronics

OK, we lied, computers don’t even understand numbers, they are really only collections of electronic switches that can be on and off. But if you think of on as 1 and off as 0 then you get simple numbers. You can then make longer numbers out of more switches so ON ON ON OFF ON = 11101 etc.

You might recognise this as Binary numbers and in GCSE or A level you get to play with these numbers rather than our normal denary numbers. In the same way that clever programmers write complex programs to turn numbers into words, sounds and pictures they also write programs that pretend that computers can understand our numbers and not just binary.

To use a computer you do not need to understand binary or even know how to write programs, but in the same way as it is useful and important to know what is beneath the engine of a car and why it needs petrol you should know a bit about computers.

The Fetch Execute cycle

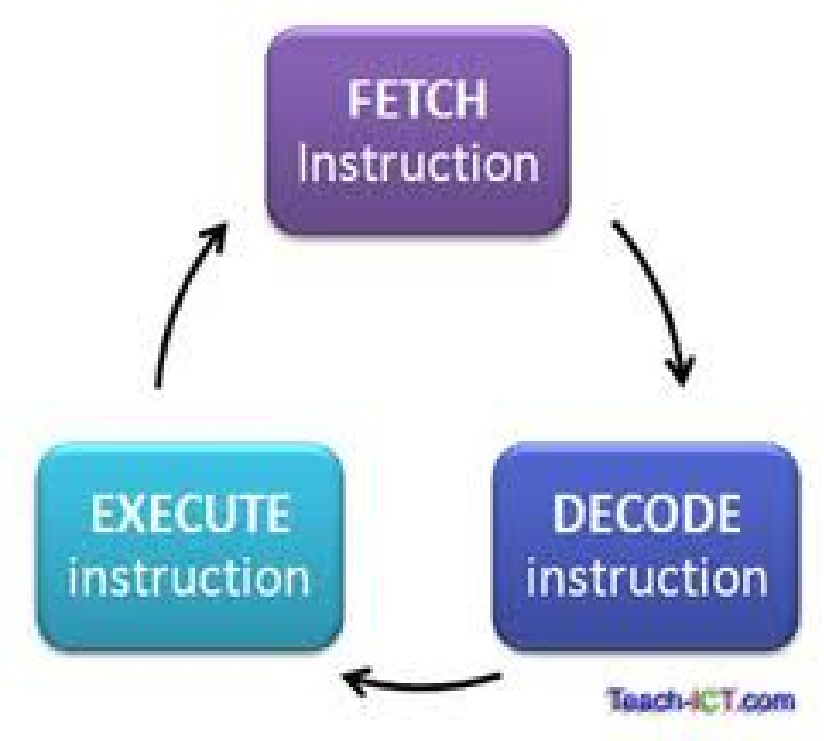
Computers are very simple machines. You turn them on and they immediately start to run a program. They then carry on running the program until you turn them off.

In fact it is a bit simpler than that. When you turn on a computer it begins to run a simple cycle where:

Get an instruction from memory
Work out what that instruction does
Carry out that instruction
Go back to fetch the next instruction etc.

These stages are given names

FETCH (Get an instruction from memory)
DECODE (Work out what that instruction does)
EXECUTE (Carry out that instruction)



The diagram above shows this very clearly and we acknowledge our colleagues at www.teach-ict.com who drew it.

In most computers there are a number of different kinds of instructions that do different kinds of things. Also most computers have instructions that simple enough to be given as just one number and other ones that need more numbers to complete the whole instruction (almost a sentence) For example an instruction 05 might mean Put x in the accumulator and the computer “intelligence” will know it needs to get the next number to work out what x is. Babbage lets you try both types of instructions.

3. What instructions can Babbage carry out?

Babbage is a very simple computer. It can only work with numbers between 00 and 99 and its memory only has 100 locations (00 to 99). It also can only read 4 input devices (0,1,2 and 3) and control 4 output devices (0-3). In spite of this it has quite a lot of instructions that allow you to write quite complex programs that can do quite clever things.

In the current version of Babbage there are 31 different instructions that Babbage can carry out. These can be divided into 4 different kinds of tasks:

- Group 1 instructions let you (Babbage really) read numbers from any memory location or from the accumulator and move them around. By using these instructions you get get a number from memory, do some arithmetic on it, perhaps with other numbers from memory and then put the answer back somewhere else.
-
- Groups 2 instructions let you do arithmetic on the number in the accumulator. In Babbage you can only carry out addition and subtraction up to 99 or down to 00.
-
- Group 3 instructions controls the 4 output devices and reads the 4 input devices. You can make decisions based on the values of the input devices and choose to do different parts of a program based on the value.
-
- Group 4 instructions let Babbage do instructions in orders different from just in sequence. You can use these instructions to write small subprograms and decide when to do them. For more experienced users (perhaps at GCSE or A level) there are stack instructions too. Finally there is a STOP instruction that is used to tell Babbage when it reaches the end of a program.

Appendix 2 gives a complete list of the instructions in numerical order and also grouped into the different kinds of instructions. The list is also available from the Info button on the toolbar in Babbage.

One number or two?

About half (actually 19) of the instructions in Babbage are simple enough that a single number can tell Babbage all the information it needs to carry out the instruction. For example 00 = Stop. Also 04 = turn on output 0.

The other half (actually 12) instructions need to give Babbage a second number before it can carry out the instruction. For example instruction 21 tells Babbage to put a number in the accumulator. Babbage expects the number to be in the memory location after the instruction 21. What this means is that Babbage knows that 21 in one location and 06 in the next location means add 06 to the accumulator.

Babbage knows which instructions only needs one number and which need two. When it fetches an instruction that needs a second number it automatically goes and gets that number too.

4. How to enter programs into Babbage

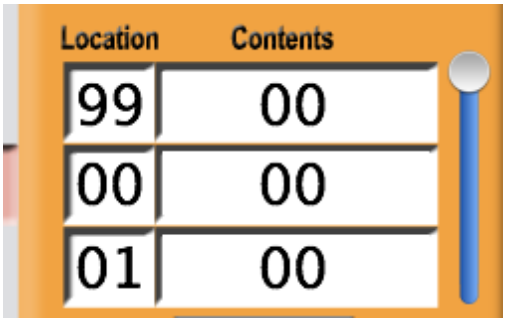
Entering programs into Babbage is a very simple task. We follow the usual digital computer rules about programs.

1. All programs and the data they use are kept in the computer's memory unit. It is up to you (the programmer) to remember where you put them.
2. Although there is no rule about it, usually programs start at location 00 and work up in sequence.
3. You need to work out in advance which instructions only need one number and which need two locations.
4. Once all of the program is in the memory you need to point the program counter to the location where your program starts (00 but could be anywhere).

To carry out these tasks and put a program into Babbage you need to use the various steppers and sliders available.

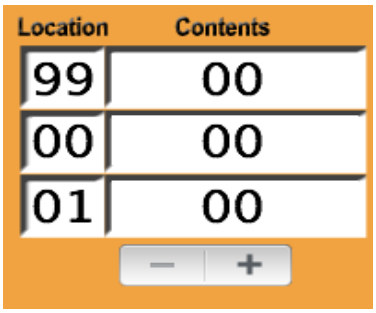
1. Write your program down on one of the blank program planning sheets (appendix 4) if it is complex.

2. Move the **memory address** register in the **control unit** to the number of the memory location you want to put your first instruction into. (you can also use the memory slider to do the same thing).



3. This will alter the window of the memory unit to show the location you have placed into the Memory unit.

4. Now use the stepper on the **Memory unit** to alter the value in the middle memory location by pressing + or – until you get the value you want. Note that for big numbers it is sometimes quicker to go down below 00 to get to it!!



5. Repeat steps 2 to 4 for the rest of your program. Make sure there is a stop instruction at the end of your program (a 00).

5. How to run programs in Babbage

You have two choices to run programs in Babbage. The difference between them is the amount of time it will take for Babbage to run your program.

Fetch Execute

In this method Babbage fetches one instruction (when you press **Fetch** on the **remote**) and Execute it when you press **Execute** on the **remote**. By repeating this process from the start of your program to the end you will carry out all of your instructions.

Using this method Babbage will show you the numbers move between different registers along the bus while it fetches and executes your instruction. It will also give you some information about the various steps it is carrying out as it does it (in the message bar at the bottom of the screen).

Usually this method is used when you first start Babbage or are learning about Fetch and Execute or even sometimes when your program goes wrong and you want to see where it goes wrong (called debugging your program).

Run Stop

When you have quite a long program (usually more than about 6 instructions) it gets a bit boring watching all of the fetch then execute stages. Also when you know how Fetch execute cycle works you dont need to be reminded all the time about the steps. When this happens you can just run the program by pressing the **Run** key on the Babbage **remote**.

If the program gets stuck or goes wrong then you will want/need the **Stop** button on the **remote** which interrupts Babbage no matter where it is in your program.

Rewind

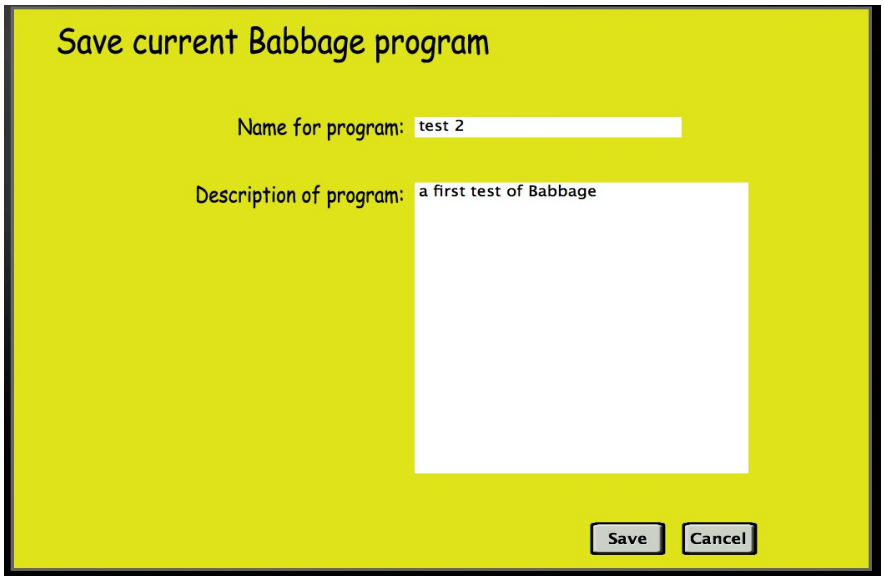
With either method it is worth remembering the Rewind button, which rewinds back to 00 and usually lets you re-run your program.

5. How to load and save programs

Babbage programs are very small and very simple, but after you have entered a program with only 5 or more instructions you will want to save your program so that you dont need to waste time next time you want to run it.

The save and load screens in Babbage are simple and the images below show how the screen will look in each.

Save program



Load Program

Note that in Save you must select one of the saved programs in the list on the left hand side by tapping them, then if you have the correct program tap Load.

Some example programs

Remember that in general programs start at location 00 in memory and follow in sequence through locations. Also remember that some instructions need to be followed with the number needed for that instruction (the data).

Name		Lights on.bab
Description		This program just turns on the outputs one after another until all are on.
Location	Instruction	Decode
00	03	Turn On output 0
01	04	Turn on Output 1
02	05	Turn on Output 2
03	06	Turn on Output 3
04	00	Stop program

Program name: cycling.bab

Name		Cycling.bab
Description		This program turns on outputs one after another, off after on and then recycles back to all off.
Location	Instruction	Decode
00	03	Turn on output 0
01	07	Turn off output 0
02	04	Turn on output 1
03	08	Turn off output 1
04	05	Turn on output 2
05	09	Turn off output 2
06	06	Turn on output 3
07	10	Turn off output 3
8	05	Turn on output 2
9	09	Turn off output 2
10	04	Turn on output 1
11	08	Turn off output 1
12	03	Turn on output 0
13	07	Turn off output 0
14	00	Stop program

Name		Add two1.bab
Description		This program adds two numbers
Location	Instruction	Decode
00	22	Put x in the accumulator
01	09	x
02	24	Add x to the accumulator
03	08	x
04	00	Stop program

Name		Add two2.bab
Description		This program adds two numbers stored in locations 30 and 31 and puts the answer in location 40
Location	Instruction	Decode
00	22	Put number from location x into accumulator
01	30	x
02	25	Add number from location x to the number in the accumulator
03	31	x
04	23	Put the number in the accumulator into the location x
05	40	x
06	Stop	
30	09	Data – the first number
31	08	Data – the second number
40	00	Data – the answer will be put here.

Appendices

Babbage instruction set
Limitations in Babbage
Some online resources
Blank Babbage program planner

Babbage Instruction Set

Instruction set categorised by opcode

opode	Mnemonic	Description
0	STOP	Stop program running
1	Inc Acc	Add 1 to the contents of the accumulator
2	Dec Acc	Subtract 1 from the contents of the accumulator
3	ON 0	Turn on Output 0
4	ON 1	Turn on Output 1
5	ON 2	Turn on Output 2
6	ON 3	Turn on Output 3
7	OFF 0	Turn off Output 0
8	OFF 1	Turn off Output 1
9	OFF 2	Turn off Output 2
10	OFF 3	Turn off Output 3
11	OFF Acc	Turn off the Output from the value of the accumulator
12	ON Acc	Turn on the Output with the value of the accumulator
13	IN 0	If Input 0 is on put 1 in accumulator otherwise put 0
14	IN 1	If Input 1 is on put 1 in accumulator otherwise put 0
15	IN 2	If Input 2 is on put 1 in accumulator otherwise put 0
16	IN 3	If Input 3 is on put 1 in accumulator otherwise put 0
17	PUSH	Copy the Program Counter to the Stack
18	POP	Retrieve the Program Counter from the Stack
19	OFF x	Turn off Output x
20	ON x	Turn on Output x
21	X > Acc	Put x in accumulator
22	(x)>Acc	Copy the contents of memory location x to the accumulator
23	Acc>(x)	Copy the accumulator to memory location x
24	Add x	Add x to the accumulator
25	Add (x)	Add the contents of memory location x to the accumulator
26	Sub x	Subtract x from the accumulator
27	Sub (x)	Subtract the contents of memory location x from the accumulator
28	GOTO x	Goto the instruction at location x next
29	Acc 0 x	If the accumulator = 0 then goto location x next
30	Acc 1 x	If the accumulator is not 0 then goto location x next

Note that instructions with x require data to be collected after opcode.

1. Instructions that move data around

21	X > Acc	Put x in accumulator
22	(x)>Acc	Copy the contents of memory location x to the accumulator
23	Acc>(x)	Copy the accumulator to memory location x

2. Instructions that carry out arithmetic

1	Inc Acc	Add 1 to the contents of the accumulator
2	Dec Acc	Subtract 1 from the contents of the accumulator
24	Add x	Add x to the accumulator
25	Add (x)	Add the contents of memory location x to the accumulator
26	Sub x	Subtract x from the accumulator
27	Sub (x)	Subtract the contents of memory location x from the accumulator

3. Instructions that change the flow of the program

17	PUSH	Copy the Program Counter to the Stack
18	POP	Retrieve the Program Counter from the Stack
28	GOTO x	Goto the instruction at location x next
29	Acc 0 x	If the accumulator = 0 then goto location x next
30	Acc 1 x	If the accumulator is not 0 then goto location x next

Note re Push and Pop.

Both of these operations use a hidden register, the stack pointer which points to the top of the stack and is decremented on push and incremented on pop (i.e. the stack grows downwards). The stack pointer (and thus the stack) is limited to between 60 and 90 to protect the user's programs against unintentional intrusion.

4. Instructions that involve input or output

3	ON 0	Turn on Output 0
4	ON 1	Turn on Output 1
5	ON 2	Turn on Output 2
6	ON 3	Turn on Output 3
7	OFF 0	Turn off Output 0
8	OFF 1	Turn off Output 1
9	OFF 2	Turn off Output 2
10	OFF 3	Turn off Output 3
11	OFF Acc	Turn off the Output from the value of the accumulator
12	ON Acc	Turn on the Output with the value of the accumulator
13	IN 0	If Input 0 is on put 1 in accumulator otherwise put 0
14	IN 1	If Input 1 is on put 1 in accumulator otherwise put 0
15	IN 2	If Input 2 is on put 1 in accumulator otherwise put 0
16	IN 3	If Input 3 is on put 1 in accumulator otherwise put 0
19	OFF x	Turn off Output x
20	ON x	Turn on Output x

Limitations in Babbage

The current version of Babbage 0.91 is the first beta release for testing on IOS devices. The next version 0.92 will test on android devices.

Limitations in the current version are:

1. all known bugs have been removed from simulator tests but there will remain some bugs, major and minor.
2. There are no included example files with the app, except on the written documentation provided in this guide.
3. Loading and saving is only onto the device's sandbox and there is no way of saving these outside the sandbox.
4. Current designs are very non IOS user interface. This is deliberate but may change, particularly if Apple and Google require it. Particularly Babbage is very non IOS 7, this is deliberate.
5. Babbage and this guide have been designed against the new English National Curriculum scheme of work and guidance. The principles of the program were written back in 1984 and are portable to any computing/it/ict curriculum that teaches students about the underlying basis of computing and not just high level code.
6. This version of Babbage was written to address KS3 and KS4 learning outcomes and many ideas and implementations and certainly this guide are not easily usable at primary levels (but we would be happy to discuss changes that would make the main principles of Babbage usable for students and teachers in these stages of education.
    1. For the record the main principles, which we believe need to be retained in any useful articulation of the underlying principles of computer operations are:
        1. instructions are seen to be numbers and not words
        2. the fetch execute cycle works at a stepping level
        3. program instructions and data are both kept in a separate memory store..
        4. the instructions carried out are simple arithmetic and movement.

Some online resources

To be completed but to include links to

National Curriculum
Computers at School group resources
Naace and other organisation resources
Examination board schemes of work and specifications.
Other resources as identified and recommended which reference school level

Kilkiesoft website materials for Babbage. www.kilkiesoft.com (in development, to be released 1 January 2014).

Online details were correct at time of publishing, please contact me martin.kilkie@kilkiesoft.co.uk if any entry is dead or incorrect.

Resource	Location
Computing ITT and CPD	https://sites.google.com/site/primaryictitt/
Computing at School, the community	
Naace	
OCR	
Edexcel	
AQA	
Teach ICT	

Blank Babbage program planner

Program Name		
Description		
Author and date		
Location	**Instruction**	**Decode**